Surface intersection using affine arithmetic

Luiz Henrique de Figueiredo
Computer Systems Group, Department of Computer Science, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
lhf@csg.uwaterloo.ca

Abstract

We describe a variant of a domain decomposition method proposed by Gleicher and Kass for intersecting and trimming parametric surfaces. Instead of using interval arithmetic to guide the decomposition, the variant described here uses affine arithmetic, a tool recently proposed for range analysis. Affine arithmetic is similar to standard interval arithmetic, but takes into account correlations between operands and sub-formulas, generally providing much tighter bounds for the computed quantities. As a consequence, the quadtree domain decompositions are much smaller and the intersection algorithm runs faster.

Keywords: Surface intersection, Range analysis, Interval arithmetic, Affine arithmetic, Parametric surfaces, CAGD.

1 Introduction

Parametric surfaces are the most popular primitives used in computer aided geometric design (CAGD). They are easy to approximate and render, and there is a huge literature on special classes of surfaces suitable for shape design, such as Bézier and splines surfaces, for which special algorithms exist [1]. However, using parametric surfaces for modeling solids in CSG systems requires efficient and robust methods for computing surface intersection, mainly for trimming surfaces into patches that can be sewn together to bound complex shapes.

Several methods have been proposed for solving the important problem of computing the intersection of two parametric surfaces. These methods can be classified into two major classes: *continuation methods* and *decomposition methods*. In this paper, we describe a variant of a decomposition method proposed by Gleicher and Kass [2]. Instead of using interval arithmetic to guide the recursive domain decomposition, as they did, this variant uses *affine arithmetic*. Affine arithmetic is similar to standard interval arithmetic, but takes into account correlations between operands and sub-formulas, generally providing much tighter bounds for the computed quantities. In many computer graphics methods based on interval arithmetic, affine arithmetic can transparently replace interval arithmetic. Variants based on affine arithmetic will probably be more efficient, but each case requires separate investigation [3]. This paper describes such an investigation for the surface intersection method by Gleicher and Kass [2].

In Section 2, we review some general methods for surface intersection. The most reliable of those seems to be recursive subdivision of parameter space based on *range analysis*, i.e., on estimates for the range of values taken by a function on subsets of its domain. Interval arithmetic is the natural technique for range analysis [4]. However, as we point out in Section 3, the excessive conservatism of interval arithmetic may greatly reduce the efficiency of the decomposition. In Section 4, we briefly describe *affine arithmetic*, a recent technique for range analysis that generally provides much tighter bounds than interval arithmetic [5]. In Section 5, we review the surface intersection algorithm proposed by Gleicher and Kass [2] and give some evidence that it can be improved by replacing interval arithmetic with affine arithmetic, specially when applied to surfaces commonly used in CAGD.

2 Previous work

Continuation methods, also called *marching methods*, use a local approach to the surface intersection problem. Starting from a point known to be on both surfaces, these methods build an approximation for the intersection curve by marching along the curve, successively computing a new point based on the previous point (or points) [6]. Continuation methods must use numerical approximations not only for marching along the curve, but also for finding starting points. Since the intersection curve may have several connected components, a starting point is needed on each component. Moreover, care must be taken for handling closed components correctly. In some applications, such as trimming, intersection curves computed with continuation methods must be mapped back to the parameter domains to define trimming curves. This may be a difficult inverse problem.

Decomposition methods, on the other hand, use a more global approach to the problem. A simple decomposition method is to build polygonal approximations for both surfaces and then intersect the corresponding polyhedral surfaces. Although it is easy to build polygonal approximations for parametric surfaces, such approximations need to be very fine to provide a good approximation for the intersection. A naive polygonal approximation is obtained by simply subdividing the parameter domain uniformly into many small rectangles. However, intersecting such fine polygonal approximation is itself a difficult task. Even if we do not care about geometric degeneracies [7, 8], this is a high complexity task: If there are n rectangles along each main direction in parameter space, then there are n^2 faces in each polyhedron. A naive algorithm that computes the intersection of the two polyhedra by testing every possible pair of faces has to consider n^4 cases, most of which do not contribute to the intersection. This algorithm is not practical because it is very expensive to refine an approximation.

Adaptive decomposition methods avoid the cost of uniform decompositions by subdividing the domain until the surface is approximately planar. In that way, the associated polygonal approximation is adapted to the local curvature of the surface, being finer in regions of high curvature and coarser in regions of low curvature, where the surface is almost flat. Such methods are generally restricted to specific types of surfaces, whose nature can be exploited to derive efficient tests for local flatness [9].

The decomposition method proposed by Gleicher and Kass [2] takes a global approach for subdividing the domains, using *range analysis* [4]. Given a rectangle in each domain, they compute an estimate for the range of values taken by the corresponding parametric function on each rectangle. This estimate is a bounding box for a surface patch, i.e., a rectangular box in 3d space, aligned with the coordinate axes, and guaranteed to contain the piece of the surface corresponding to the given rectangle in parameter space. If two bounding boxes do not intersect, then the corresponding surfaces patches cannot intersect. If the bounding boxes do intersect, then the surfaces patches *may* intersect. In this case, the rectangles are subdivided, and the process is repeated until either the surfaces patches are proved disjoint or a user defined tolerance is reached; the patches are then assumed to intersect. Gleicher and Kass use interval arithmetic for computing ranges. In this paper, we show that their method can be improved by replacing interval arithmetic with affine arithmetic, a tool recently introduced for range analysis that generally produces better bounds than interval arithmetic [5].

Since decomposition methods work directly on parameter domains, no inverse problem needs to be solved to find trimming curves. On the other hand, decomposition methods compute trimming curves in a piecewise, unstructured way; the pieces must be somehow glued together into complete curves. In addition to the domain decomposition method for finding intersections with interval arithmetic, Gleicher and Kass [2] also propose complementary algorithms for finding trimming curves and triangulating the domains to define trimmed surfaces. These algorithms do not depend on range analysis and therefore can be applied to the decompositions computed by the variant presented here. For this reason, we concentrate on showing that their algorithm can be improved by using affine arithmetic instead of interval arithmetic.

3 Interval arithmetic

The classical technique of *interval arithmetic* (IA) provides a natural tool for range analysis [4]. In IA, each quantity is represented by an interval of floating-point numbers. Those intervals are added, subtracted, multiplied, etc., in such a way that each computed interval is guaranteed to contain the (unknown) value of the quantity it represents.

Simple formulas are easily derived for performing the primitive arithmetic operations on intervals. Interval extensions for a complicated function can be computed by composing these primitive formulas in the same way the primitive operations are composed to compute the function itself. In other words, any algorithm for computing a function using primitive operations can be readily (and automatically) interpreted as an algorithm for computing an interval extension for the same function. This is specially elegant to implement with programming languages that support operator overloading, such as C++, Ada, Pascal-SC and Fortran-90, but can be easily implemented in any programming language, either manually or with the aid of a pre-compiler. Since it is also relatively easy to provide interval extensions for elementary transcendental functions such as sin, cos, log, and exp, the class of functions for which interval extensions can be easily (and automatically) computed is much larger than the class of rational polynomial functions.

Several methods based on IA have recently been proposed for solving fundamental problems in computer graphics, such as ray tracing [10] and the approximation of implicit surfaces [11, 12, 13]. Those methods have become quite popular, due to their

ability to handle arbitrarily complex non-polynomial surfaces, and their immunity to round-off errors.

Previously, methods based on Lipschitz conditions (global bounds on derivatives) appeared to be promising for computer graphics applications [9, 14]. However, computing Lipschitz bounds is a non-trivial mathematical problem that did not seem to have an automatic solution. Methods using range analysis seem to be more popular now in computer graphics, specially because range analysis can be automated (typically with IA) [4]. In particular, Lipschitz bounds can be computed using automatic differentiation and interval arithmetic [15]. Global optimization, which includes computing Lipschitz bounds as a special case, has recently been shown to be feasible with range analysis [16, 17]. However, global optimization with range analysis has barely been explored in computer graphics [12].

The main weakness of IA is that it tends to be too conservative: the computed interval for a quantity may be much wider than the exact range of that quantity, often to the point of uselessness. This over-conservatism is mainly due to the assumption that the (unknown) values of the arguments to primitive operations may vary *independently* over the given interval. If there are any mathematical constraints between these arguments, then not all combinations of values in the corresponding intervals will be valid. As a consequence, the result interval computed by IA may be much wider than the exact range of the result quantity. This is sometimes called the *dependency problem* in IA.

As an example of how dependencies are overlooked in IA, consider evaluating $x(10-x)$, where x is known to lie in the interval $\bar{x} = [4\ ..\ 6]$. Applying the IA formulas blindly, we get:

$$\begin{aligned} \bar{x} &= [4\ ..\ 6] \\ 10 - \bar{x} &= [10\ ..\ 10] - [4\ ..\ 6] = [4\ ..\ 6] \\ \bar{x}(10 - \bar{x}) &= [4\ ..\ 6] \cdot [4\ ..\ 6] = [16\ ..\ 36], \end{aligned}$$

which is 20 times wider than the exact range of the expression $x(10 - x)$ over $[4\ ..\ 6]$, namely $[24\ ..\ 25]$. The large discrepancy between the two intervals is due to the inverse relation between the quantities x and $10 - x$, which is not known to the IA multiplication algorithm. Inverse relations such as this are common in curve and surface parametrizations used in CAGD, as the examples in Section 5 show.

The over-conservatism of IA is particularly bad in long computation chains, where the intervals computed by one stage of the chain are the inputs to the following stage. In such cases, one often observes an "error explosion": as the evaluation advances down the chain, the relative accuracy of the computed intervals decreases exponentially, and they soon become too wide to be useful, by many orders of magnitude. Unfortunately, long computations chains are not uncommon in computer graphics applications.

4 Affine arithmetic

Affine arithmetic (AA) is a model for numerical computation recently proposed to address the "error explosion" problem in IA [5]. Like IA, affine arithmetic keeps track automatically of the round-off and truncation errors affecting each computed quantity. Unlike IA, however, AA keeps track of *correlations* between those quantities. This extra information allows AA to provide much tighter range estimates than IA, especially in long computation chains.

The key feature of AA is an extended encoding of quantities from which one can determine, in addition to their ranges, also certain relationships to other quantities — such as the ones existing between x and $10-x$ in the example in Section 3. Specifically, a partially unknown quantity x is represented in AA by an *affine form* $\hat{x}$, which is a first-degree polynomial:

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n.$$

Here, the x_i are known real coefficients (stored as floating-point numbers), and the ε_i are symbolic variables, called *noise symbols*, whose values are unknown but assumed to lie in the interval $[-1\ ..\ +1]$. Noise symbols stand for independent sources of error or uncertainty that contribute to the total uncertainty of the quantity x; the coefficient x_i gives the magnitude of that contribution for the source ε_i.

The main benefit of encoding quantities with affine forms instead of intervals is that the same noise symbol ε_i may contribute to the uncertainty of two or more quantities (inputs, outputs, or intermediate results) arising in the evaluation of an expression. The sharing of a noise symbol ε_i by two affine forms $\hat{x}$, $\hat{y}$ indicates a partial dependency between the underlying quantities x, y. The magnitude and sign of the dependency is determined by the corresponding coefficients x_i, y_i. Taking such correlations into account allows better range estimates to be computed (see the example at the end of this section).

Other approaches to the dependency problem in IA include *centered forms* [4] and Hansen's *generalized interval arithmetic* [18], in which quantities are represented by affine combinations of a *fixed* number of *intervals*. In AA, new noise symbols are dynamically created during a long computation.

As one may expect, affine arithmetic is more complex and expensive than ordinary interval arithmetic. However, its higher accuracy is worth the extra cost in many computer graphics applications, including adaptive enumeration of implicit objects [3] and computing the intersection of parametric surfaces, as we show in Section 5.

The use of AA for range analysis is simple: First convert all input intervals to affine forms. Then operate on these affine forms with AA to compute the desired function. Finally, convert the result back into an interval.

The conversion steps are simple. Given an interval $\bar{x} = [a\,..\,b]$ representing some quantity x, an equivalent affine form for the same quantity is given by $\hat{x} = x_0 + x_k\varepsilon_k$, where

$$x_0 = \frac{b+a}{2} \quad \text{and} \quad x_k = \frac{b-a}{2}\ .$$

Since input intervals are assumed to be unrelated, because they usually represent independent variables, a new noise symbol ε_k must be used for each input interval.

Conversely, the value of a quantity represented by an affine form $\hat{x} = x_0+x_1\varepsilon_1+\cdots+x_n\varepsilon_n$ is guaranteed to be in the interval

$$[\hat{x}] = [x_0 - \xi\,..\,x_0 + \xi],\ \text{where}\ \xi = \|\hat{x}\| := \sum_{i=1}^{n} |x_i|\,.$$

Note that $[\hat{x}]$ is the smallest interval that contains all possible values of $\hat{x}$, assuming that each ε_i ranges independently over the interval $[-1\,..\,+1]$.

Computing with affine arithmetic

To evaluate a formula in AA, we must replace each of its elementary operations $z \leftarrow f(x,y)$ on real numbers by an equivalent operation $\hat{z} \leftarrow \hat{f}(\hat{x},\hat{y})$ on affine forms, where $\hat{f}$ is a procedure that computes an affine form for $z = f(x,y)$ that is consistent with $\hat{x}$, $\hat{y}$.

When f is an affine function of x, y, the value $\hat{z}$ can be expressed exactly as an affine combination of the noise symbols ε_i. More precisely, if

$$\begin{aligned}\hat{x} &= x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n\\ \hat{y} &= y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n,\end{aligned}$$

and $\alpha \in \mathbf{R}$, then

$$\begin{aligned}\hat{x} \pm \hat{y} &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n\\ \alpha\hat{x} &= (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n\\ \hat{x} \pm \alpha &= (x_0 \pm \alpha) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n.\end{aligned}$$

Note that, according to those formulas, the difference $\hat{x} - \hat{x}$ between an affine form and itself is identically zero. In this case, the fact that the two operands share the same noise symbols with the same coefficients reveals that they are actually the same quantity, and not just two quantities that happen to have the same range of possible values. Thanks to this feature, in AA we also have $(\hat{x} + \hat{y}) - \hat{x} = \hat{y}$, $(3\hat{x}) - \hat{x} = 2\hat{x}$, and so on. Such properties are *not* valid in IA, and are one source of error explosion.

When f is not an affine operation, the value $\hat{z}$ cannot be expressed exactly as an affine combination of the ε_i. In that case, we pick the best affine approximation to f (best in the Chebyshev sense of minimizing the maximum error), and then append an extra term $z_k\varepsilon_k$ to represent the error introduced by this approximation:

$$\hat{z} = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n + z_k\varepsilon_k.$$

Here, ε_k must be a brand new noise symbol (i.e., distinct from all other noise symbols in the same computation) and z_k must be an upper bound for the approximation error. Note that, unlike Hansen's generalized interval arithmetic [18], new noise symbols are created during a long AA computation. They account for extra sources of uncertainty introduced during the computation, such as approximation errors and round-off errors.

Using this approach, formulas can be derived for all elementary operations and functions, both algebraic and transcendental. For example, the multiplication of two affine forms $\hat{x}$, $\hat{y}$ is given by

$$\begin{aligned}z_0 &= x_0y_0\\ z_i &= x_0y_i + y_0x_i \qquad (i = 1..n)\\ z_k &= \|\hat{x}\|\,\|\hat{y}\|.\end{aligned}$$

Like Lipschitz bounds, Chebyshev approximations must be computed by hand. Unlike Lipschitz bounds, however, Chebyshev approximations need to be found only for primitive functions because AA formulas for primitive functions can be automatically combined into formulas for arbitrarily complex functions, as described in Section 3 for IA.

To see how AA handles the dependency problem, consider again evaluating $z = x(10 - x)$, for x in the interval $[4\,..\,6]$, but now using AA instead of IA:

$$\begin{aligned}\hat{x} &= 5 + 1\varepsilon_1\\ 10 - \hat{x} &= 5 - 1\varepsilon_1\\ \hat{z} = \hat{x}(10 - \hat{x}) &= 25 + 0\varepsilon_1 - 1\varepsilon_2\\ [\hat{z}] &= [25 - 1\,..\,25 + 1] = [24\,..\,26].\end{aligned}$$

Observe that the influence of the noise symbol ε_1 in the factors happened to cancel out (to first order) in the product. Note also that the range of $\hat{z}$ is much closer to [24 .. 25], the exact range of z, and much better than the IA estimate, [16 .. 36].

5 Examples

In this section, we show two examples of how the Gleicher-Kass algorithm for surface intersection [2] can be improved by using AA instead of IA, specially for surfaces that are common in CAGD.

Recall that their algorithm is a domain decomposition algorithm that uses range analysis to decide whether two surfaces patches intersect. If the bounding box estimates provided by range analysis for the patches do not intersect, then the patches cannot intersect. If the bounding boxes do intersect, then the surfaces patches may intersect, and the corresponding rectangles in the domains are subdivided into four equal pieces and further tested. In this way, a quadtree decomposition is built for each domain. For efficiency, Gleicher and Kass keep track of all pairs of patches that might intersect: each leaf node in one quadtree contains a list of leaf nodes in the other quadtree that it overlaps. This list is refined and distributed to its children when a node is subdivided. The main step in the algorithm is the subdivision of a leaf node [2]:

```
subdivide(n):
    if n's overlap list is not empty
        subdivide n into four children
        for each i in n's overlap list
            remove n from i's overlap list
            for each child c of n
                if c overlaps i
                    add i to c's list
                    add c to i's list
```

Gleicher and Kass [2] remark that this subdivision step can be applied in several different orders. They actually combine depth-first search with breadth-first search to control the size and accuracy of the sampling of the intersection curve. Our simple implementation uses only breadth-first search, by enqueueing new nodes as they are created and then subdividing a node from the queue at a time. For efficiency, a bounding box for a node is computed exactly once, when the node is created. This happens in each subdivision, when four nodes are created, and also at startup, when one node for each entire domain is created. Note that the use of range analysis is restricted to the computation of bounding boxes, and this depends exclusively on one surface at a time. The examples below show how this algorithm performs when bounding boxes are computed with IA and with AA.

5.1 Lofted parabolas

Consider a cubic patch obtained by lofting a parabola to another parabola. More precisely, take three points a_0, a_1, a_2 in $\mathbf{R}^3$, and consider the quadratic Bézier curve defined by these points:

$$\alpha(u) = a_0(1-u)^2 + 2a_1u(1-u) + a_2u^2,$$

for $u \in [0,1]$. Take three other points b_0, b_1, b_2 in $\mathbf{R}^3$, and the Bézier parabola defined by them:

$$\beta(u) = b_0(1-u)^2 + 2b_1u(1-u) + b_2u^2,$$

for $u \in [0,1]$. Now, sweep α to β linearly to obtain a surface:

$$f(u,v) = (1-v)\alpha(u) + v\beta(u),$$

for $u, v \in [0,1]$. Lofting is a common operation in CAGD. Figure 1 shows two intersecting lofted parabolas (skew parabolic cylinders in this case).

Because the parametrization f contains several occurrences of u and $1-u$, and of v and $1-v$, the terms are strongly correlated, and we expect AA to provide tighter bounds for f than IA. This expectation is met: Figure 2 shows the domain decompositions built with IA and AA for computing the intersection of the two lofted parabolas shown in Figure 1. Both cases use six levels of recursive subdivision. In the decomposition based on IA, 5314 bounding boxes were computed and 3360 patches remained as possibly intersecting. The decomposition based on AA was approximately 3 times more efficient: 1930 bounding boxes were computed and 968 patches remained. Note how AA exploits correlations to give much tighter approximations for the intersection, quickly discarding large parts of both domains. With no graphics output, the AA version ran approximately 3 times faster than the IA version. (Timings performed on a personal IBM RS6000/320 workstation with typical load.)

5.2 Bicubic patches

Consider now bicubic patches, the most common surface patches in CAGD. A bicubic patch is a tensor product Bézier surface, defined by a mesh of sixteen control points $a_{ij} \in \mathbf{R}^3$ $(i,j = 0..3)$:

$$f(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}B_i^3(u)B_j^3(v),$$

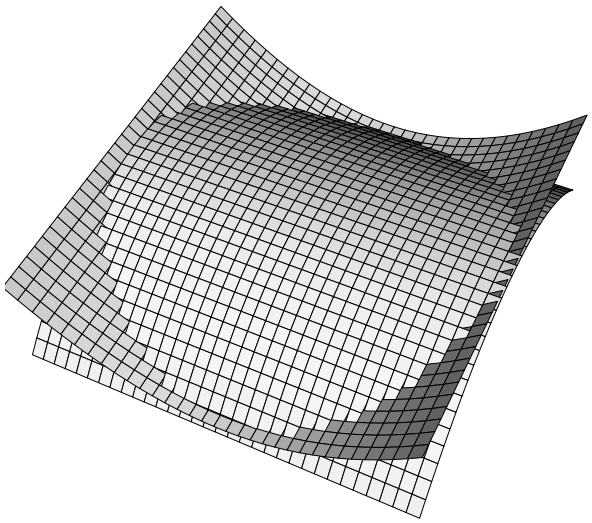

Figure 1: Two intersecting lofted parabolas.

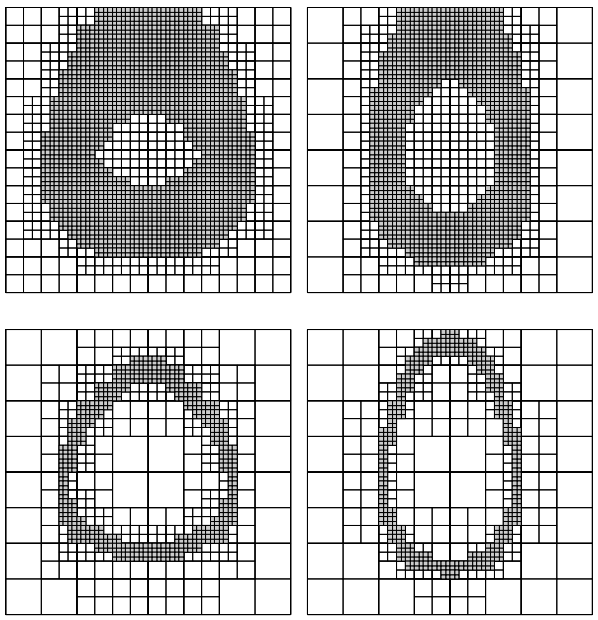

Figure 2: Domain decompositions computed with IA (top) and AA (bottom) for intersecting the two skew parabolic cylinders shown in Figure 1. Six levels of recursive subdivision were performed. The patch on the left has control points $a_0 = (0, 0, 0)$, $a_1 = (1, 0, 1)$, $a_2 = (2, 0, 0)$, $b_0 = (0, 2, 0)$, $b_1 = (1, 2, 1)$, $b_2 = (2, 2, 0)$. The patch on the right has control points $a_0 = (0, 0, 0.55)$, $a_1 = (0, 1, -0.45)$, $a_2 = (0, 2, 0.55)$, $b_0 = (2, 0, 0.55)$, $b_1 = (2, 1, -0.45)$, $b_2 = (2, 2, 0.55)$.

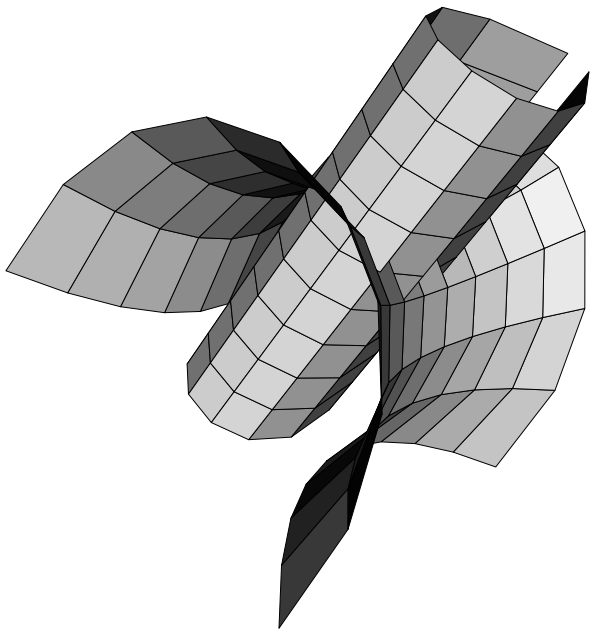

Figure 3: Two intersecting bicubic patches.

where $u, v \in [0, 1]$ and B_i^n is the i-th Bernstein polynomial of degree n:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}.$$

(The lofted parabolas in Section 5.1 are also tensor product Bézier surfaces.)

Figure 3 shows two intersecting bicubic patches. Figure 4 shows the domain decompositions built with IA and AA for computing the intersection of these two bicubic patches. Because tensor product parametrizations contain many occurrences of strongly correlated terms, we expect AA to provide tighter bounds than IA. Again, this expectation is met. In the decomposition based on IA, 8038 bounding boxes were computed and 5508 patches remained as possibly intersecting. The decomposition based on AA was much more efficient: 1786 bounding boxes were computed and 728 patches remained. Thus, AA computed approximately 4.5 times fewer bounding boxes than IA and generated an approximation 7.6 times more accurate. With no graphics output, the AA version ran approximately 3.7 times faster than the IA version.

An extra subdivision step with AA is sufficient to show that the intersection curve is not a loop (Figure 5). After this extra step, a total of 3066 bounding boxes were computed and 1280 patches remained.

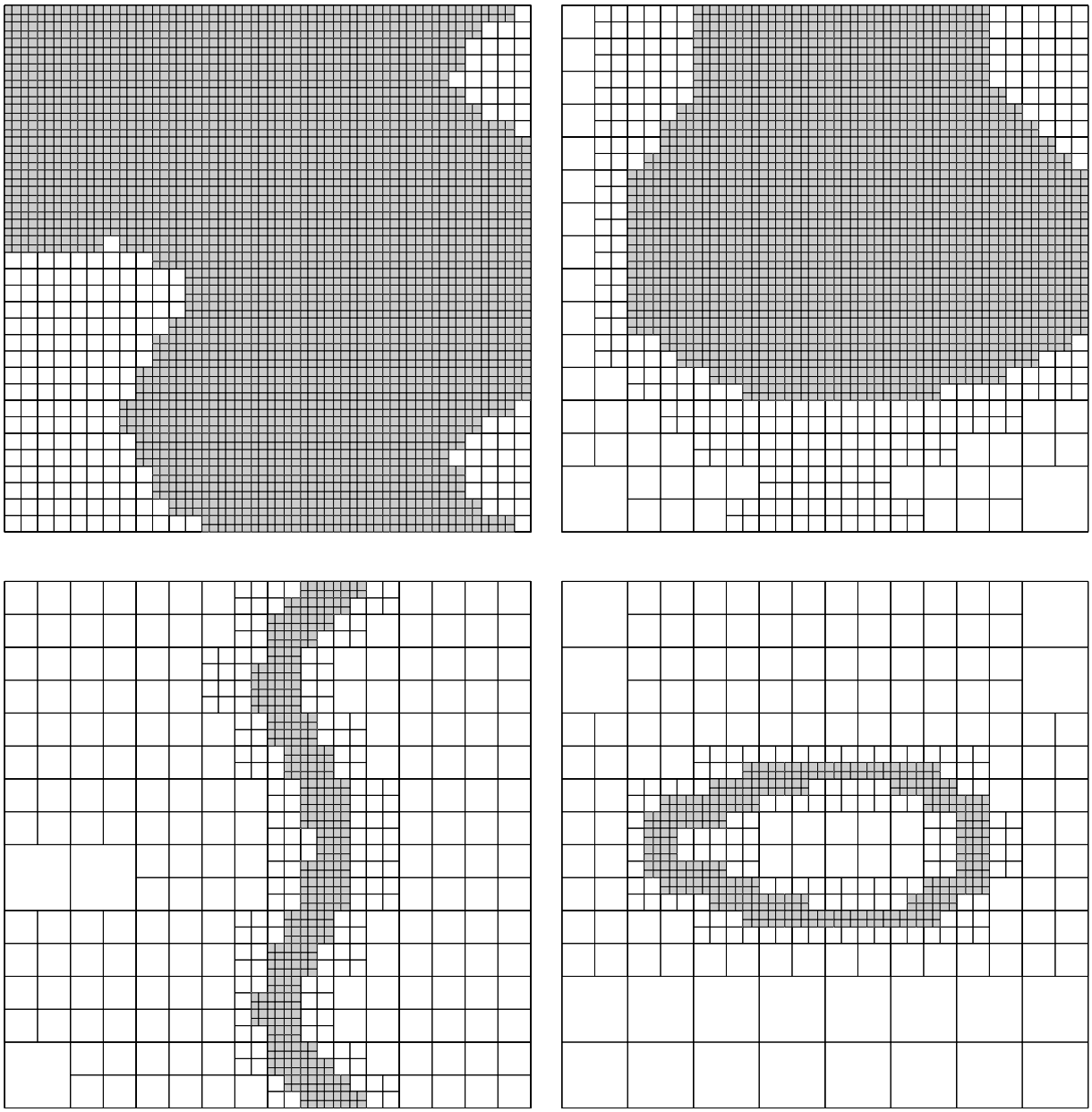

Figure 4: Domain decompositions computed with IA (top) and AA (bottom) for intersecting the two bicubic patches shown in Figure 3. Six levels of recursive subdivision were performed. The patch on the left has control points $(1.4, 0, 0.5)$, $(0, 0, 3)$, $(3, 0, 3)$, $(1.6, 0, 0.5)$, $(1.4, 1, 0.5)$, $(0, 1, 3)$, $(3, 1, 3)$, $(1.6, 1, 0.5)$, $(1.4, 2, 0.5)$, $(0, 2, 3)$, $(3, 2, 3)$, $(1.6, 2, 0.5)$, $(1.4, 3, 0.5)$, $(0, 3, 3)$, $(3, 3, 3)$, $(1.6, 3, 0.5)$. The patch on the right has control points $(0, 0, 0)$, $(0, 3, 0)$, $(3, 3, 0)$, $(3, 0, 0)$, $(1, 0, 1)$, $(0, 2, 1)$, $(3, 2, 1)$, $(2, 0, 1)$, $(1, 0, 2)$, $(0, 2, 2)$, $(3, 2, 2)$, $(2, 0, 2)$, $(0, 0, 3)$, $(0, 3, 3)$, $(3, 3, 3)$, $(3, 0, 3)$.

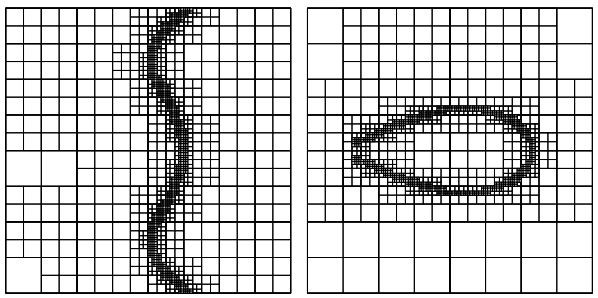

Figure 5: Extra subdivision step with AA shows that intersection curve is not a loop.

6 Conclusion

The surface intersection algorithm proposed by Gleicher and Kass [2] is robust, simple to implement, and its use of interval arithmetic is localized, making it easy to use affine arithmetic instead. Although AA is indeed more accurate than IA, it is more complex to implement and more expensive to run. However, as shown by the examples, its higher accuracy is worth the extra cost for computing the intersection of parametric surfaces, specially the surfaces commonly used in CAGD, because of the many correlations present in their parametrizations. The higher accuracy of AA translates into more efficient domain decompositions, even though primitive operations in AA are more expensive than in IA. Because the decompositions are smaller, there are fewer pairs of patches to test for intersection, and the whole algorithm runs faster.

We plan to continue to investigate computer graphics problems that have solutions based on range analysis which would benefit from replacing IA with AA. We expect variants based on AA to be more efficient, but each case requires separate investigation.

Acknowledgements. J. Stolfi provided advice on AA and some code. Figures 1 and 3 were generated with Geomview (software written at the Geometry Center, University of Minnesota, and available at `http://www.geom.umn.edu/software/`). The author holds a post-doctoral fellowship from the Brazilian Council for Scientific and Technological Development (CNPq).

References

[1] R. E. Barnhill. Surfaces in computer-aided geometric design: A survey with new results. *Computer Aided Geometric Design*, 2(1–3):1–17, 1985.

[2] M. Gleicher and M. Kass. An interval refinement technique for surface intersection. In *Proceedings of Graphics Interface '92*, pages 242–249, May 1992.

[3] L. H. de Figueiredo and J. Stolfi. Adaptive enumeration of implicit surfaces with affine arithmetic. In *Proceedings of Implicit Surfaces '95*, pages 161–170, April 1995. Extended version to appear in *Computer Graphics Forum.*

[4] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions.* Ellis Horwood Ltd., 1984.

[5] J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proceedings of VI SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing)*, pages 9–18, 1990. Available at `http://dcc.unicamp.br/~stolfi/`.

[6] R. E. Barnhill, G. Farin, M. Jordan, and B. R. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(1-2):3–16, July 1987.

[7] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann, 1989.

[8] A. J. Stewart. Local robustness and its applications to polyhedral intersection. *International Journal on Computational Geometry and Applications*, 4(1):87–118, 1994.

[9] D. Filip, R. Magedson, and R. Markot. Surface algorithms using bounds on derivatives. *Computer Aided Geometric Design*, 3(4):295–311, 1986.

[10] D. P. Mitchell. Robust ray intersection with interval arithmetic. In *Proceedings of Graphics Interface '90*, pages 68–74, May 1990.

[11] K. G. Suffern and E. D. Fackerell. Interval methods in computer graphics. *Computers & Graphics*, 15:331–340, 1991.

[12] J. M. Snyder. Interval analysis for computer graphics. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):121–130, July 1992.

[13] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):131–138, July 1992.

[14] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):297–306, July 1989.

[15] L. B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science.* Springer Verlag, 1981.

[16] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood Ltd., 1988.

[17] E. Hansen. *Global optimization using interval analysis.* Number 165 in Monographs and textbooks in pure and applied mathematics. M. Dekker, 1988.

[18] E. Hansen. A generalized interval arithmetic. In K. Nickel, editor, *Interval mathematics*, number 29 in Lecture Notes in Computer Science, pages 7–18. Springer Verlag, 1975.